Functional Magic

Tips and Tricks with Oracle7 Stored Procedures

By John C. Lennon

The impossible done immediately — miracles take a little longer. This is truly the case with Oracle7 stored functions and procedures. Their potential seems limited only by the imagination of the developer. Especially useful are stored functions that can be embedded in SQL — they help to overcome some of the shortcomings of SQL.

This article introduces a simple function (see Figure 1), and then demonstrates a variety of uses for it. The examples shown are based on the simple data model shown in Figure 2.

Ordering the Un-Orderable

Many applications use unique identifiers (UIDs) instead of natural keys. (UIDs — also known as surrogate keys — are the subject of ongoing debate, but that debate is outside the scope of this article.)

```
FUNCTION Decode_Customer_Uid(p_Uid NUMBER)
   RETURN VARCHAR2
IS
   return_value VARCHAR2(10) := ' UNKNOWN';

   CURSOR Get_Short_Name IS
      SELECT short_name
        FROM customers
       WHERE customer_uid = p_Uid;

BEGIN

     OPEN  Get_Short_Name;
    FETCH  Get_Short_Name
     INTO  return_value;
    CLOSE  Get_Short_nmae;

   RETURN  return_value;

END;
```

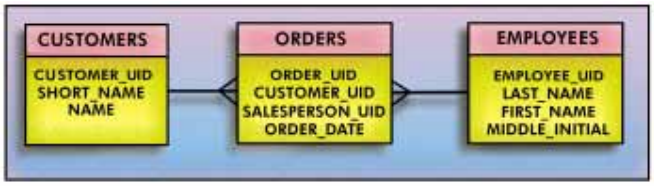

Figure 1 (Top): A simple PL/SQL stored function.
Figure 2 (Bottom): The example data model.

A disadvantage of using UIDs is that a foreign key lookup is needed to order data by the natural key. If a query can join the base table with the foreign key table, this is not a problem. However, Oracle Forms does not allow a join to be added to a block's base table query, and using a view is not always an acceptable alternative.

The query constructed by Oracle Forms for a block based on the ORDERS table in Figure 2 would be:

```
SELECT rowid, order_uid, customer_uid,
       salesperson_uid, order_date
  FROM orders
```

What we would like to append to this query, to display the data ordered by a customer, is:

```
ORDER BY SELECT  short_name
           FROM  customers
          WHERE  customer_uid = orders.-
                 customer_uid
```

This isn't legal of course, and SQL responds with "Error at line 3 ORA-00936: missing expression". Examining the function however, it's apparent that the sub-query above is exactly what it is doing. Therefore, we embed the function in the ORDER BY clause as follows:

```
ORDER BY decode_customer_uid(customer_uid)
```

This is done in the block property sheet, as shown in Figure 3.

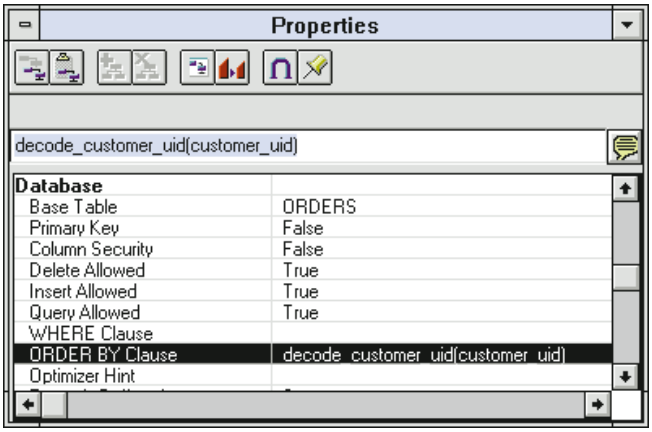

Figure 3: The Oracle Forms block property sheet.

The function returns the customer short name for each row using the customer_uid of that row as its argument. SQL sorts on this value exactly as if it were a base table column. Note that if a match is not found, the function returns "UNKNOWN". The leading space will cause such items to be returned at the head of the list.

If we want to order by salesperson instead of (or as well as) customer, we simply create another (similar) function based on the Employees table (see Figure 4).

```
FUNCTION Decode_Employee_Uid(p_Uid NUMBER)
   RETURN VARCHAR2
IS

   return_value VARCHAR2(10) := ' UNKNOWN';

   CURSOR Get_Last_Name IS
       SELECT last_name
         FROM employees
        WHERE employee_uid = p_Uid;

BEGIN

     OPEN Get_Last_Name;
    FETCH Get_Last_Name
     INTO return_value;
    CLOSE Get_Last_Name;

   RETURN return_value;

END;
```

Figure 4: A function based on the Employees table.

We can now use this function in our ORDER BY clause:

```
ORDER BY decode_employee_uid(salesperson_uid)
```

or

```
ORDER BY decode_customer_uid(customer_uid),
         decode_employee_uid(salesperson_uid)
```

Pre-Query Trigger

A second use of our function in Oracle Forms is in a pre-query trigger. The problem is similar in that the basic query is fixed, and we cannot introduce a join to restrict our query. However, we can use a sub-query to achieve this. (Note: For clarity, the quotes that would usually be found in the triggers have been omitted in the following examples.)

To include a lookup item in a query in a Designer/2000-generated form, we typically see code such as this in a pre-query trigger:

```
AND customer_uid IN (SELECT  customer_uid
                       FROM  customers
                      WHERE  name LIKE :orders.dsp_name)
```

This is not very efficient code, because it forces a full table scan of the Customers table. To improve performance it can be replaced with:

```
AND EXISTS (SELECT 1
              FROM  customers
             WHERE  name LIKE :orders.dsp_name
               AND  customer_uid = orders.customer_uid)
```

That's much better. And at least it uses the index we took so much trouble creating. Or does it? If we want to make the query case-insensitive (a common requirement), we must change it to:

```
AND EXISTS (SELECT 1
              FROM  customers
             WHERE  UPPER(name) LIKE UPPER(:orders.dsp_name)
               AND  customer_uid = orders.customer_uid)
```

and we are back to a full table scan. To overcome this, we can use our function:

```
AND UPPER(Decode_Customer_Uid(customer_uid)
    LIKE UPPER(:orders.dsp_name)
```

What does this achieve (besides a little less typing)? Considerably better performance. We are now converting our customer name to upper case for comparison after the function retrieves it. The query within the function will use the index on the customer UID to retrieve the name by rowid.

Post-Query and Post-Change Triggers

Another use for these functions is in post-query and post-change triggers.

```
SELECT short_name
  INTO :orders.dsp_name
  FROM customers
 WHERE customer_uid = :orders.customer_uid;

SELECT last_name
  INTO orders.dsp_name2
  FROM employees
 WHERE employee_uid = :orders.salesperson_uid;
```

can be replaced with:

```
:orders.dsp_name  :=
   decode_customer_uid(:orders.customer_uid);
:orders.dsp_name2 :=
   decode_employee_uid(:orders.salesperson_uid);
```

Not only is this simpler and more elegant code, but in a client/server environment, the database queries have been moved to the server, and network traffic is reduced.

CONNECT BY

A common problem is that, when querying a hierarchical table using CONNECT BY, joins are not permitted, and therefore all the required information cannot be retrieved with a single query.

For example:

```
 SELECT  a.org_name, b.last_name
   FROM  org_chart a, employees b
  WHERE  a.manager_uid = b.employee_uid
  START  WITH LEVEL = 1
CONNECT  BY a.org_uid = PRIOR a.parent_org_uid;
```

will return an error "ORA-01437: cannot have a join with CONNECT BY".

Using the function shown in Figure 4, this problem is easily resolved with the following query:

```
 SELECT  org_name, decode_employee_uid(manager_uid)
   FROM  org_chart
  START  WITH LEVEL = 1
CONNECT  BY org_uid = PRIOR parent_org_uid;
```

Packaging the Functions

It's likely there will be many of these functions in an application, and packaging them offers real benefits — most importantly performance. When an element in a package is referenced for the first time, the entire package is loaded into the system global area (SGA). Also, the effort required to create functions is reduced, because only one public synonym (for the package) is required, rather than one for each function. Also, users need only execute privileges for the package.

Figure 5 (the package specification) and Figure 6 (the package body) illustrate how functions may be packaged. Note the use of the pragma to assert the purity level for each function in the package specification. This is a necessary instruction to the compiler to guarantee that the functions do not modify any database tables (WNDS) or any package variables (WNPS). If the pragma is omitted, attempts to embed the functions in SQL will result in an error.

Moving the functions into a package changes the way in which they are called. The package must be referenced using dot notation. (i.e. `package.function`). For example:

```
ORDER BY Decode_Uid.Customer(customer_uid);
```

Conclusion

Stored functions and procedures offer a wealth of features and benefits not yet widely appreciated. They represent a move towards object-oriented implementation, and can make a significant contribution to performance gains. User-defined functions embedded in SQL can be used in the same way as built-in SQL functions, and add a degree of procedural language functionality to SQL.

It is well worth the time and effort required for any developer to learn to use these powerful and productive new tools.

```
PACKAGE Decode_Uid IS

   FUNCTION Customer (p_In_Uid NUMBER)
      RETURN VARCHAR2;
      PRAGMA RESTRICT_REFERENCES(Customer,WNDS,WNPS);

   FUNCTION Employee(p_In_Uid NUMBER
      RETURN VARCHAR2;
      PRAGMA RESTRICT_REFERENCES(Employee,WNDS,WNPS);

END Decode_Uid;
```

```
PACKAGE BODY Decode_Uid IS

   FUNCTION Customer(p_Uid NUMBER)
      RETURN VARCHAR2
   IS

      return_value VARCHAR2(10) := ' UNKNOWN';

      CURSOR Get_Short_Name IS
         SELECT short_name
           FROM customers
          WHERE customer_uid = p_Uid;

   BEGIN

       OPEN Get_Short_Name;
      FETCH Get_Short_Name
       INTO return_value;
      CLOSE Get_Short_nmae;

     RETURN return_value;

   END Customer;

   FUNCTION Employee(p_In_Uid INTEGER)
      RETURN VARCHAR2
   IS
      return_value VARCHAR2(20) := ' UNKNOWN';

      CURSOR Get_Last_Name is
         SELECT last_name
           FROM employees
          WHERE employee_uid = p_In_Uid;

   BEGIN

       OPEN Get_Last_Name
      FETCH Get_Last_Name
       INTO return_value
      CLOSE Get_Last_Name;

     RETURN return_value;

   END Employee;

END Decode_Uid;
```

Figure 5 (Top): A sample package specification.
Figure 6 (Bottom): An example package body.

John Lennon is a consultant with Utility Partners, Inc., a company specializing in software technology for utilities. He has been working with Oracle products in South Africa and the United States since 1988. John can be contacted at (702) 498-4990, Fax: (702) 871-4318, or e-mail: jomarlen@aol.com.

Available December 1996

The Must Have Reference Source For The Serious Oracle® Developer

oracle informant

CD ROM

The Entire Text of all Technical Articles Appearing in Oracle® Informant® in 1996

The Oracle® Informant® Works 1996 CD-ROM will include:

- All Technical Articles
- Text and Keyword Search Capability
- Improved Speed and Performance
- All Supporting Code and Sample Files
- 16-Page Color Booklet
- Third-Party Add-In Product Demos
- CompuServe Starter Kit with $15 Usage Credit.

A $130 Value
Available Now for only

$49.95

California residents add 7¼% Sales Tax, plus $5 shipping & handling for US orders.
(International orders add $15 shipping & handling)

Call Now Toll Free
1-800-88-INFORM

1-800-884-6367 Ask for offer # WEB
To order by mail,
send check or Money Order to:
Informant Communications Group, Inc.
ATTN: Works CD offer # WEB
10519 E. Stockton Blvd, Suite 142
Elk Grove, CA 95624-9704
or Fax your order to 916-686-8497

Get Informed!

Subscribe to Oracle Informant, The Independent Monthly Guide to Oracle Development.

Order Now and Get One Issue FREE!

For a limited time you can receive the first issue FREE plus 12 additional issues for only $49.95 That's nearly 25% off the yearly cover price!

Each big issue of *Oracle Informant* is packed with Oracle tips, techniques , news, and more!

- Client/Server Development
- Using Developer/2000™
- Tuning Oracle7
- PL/SQL Techniques
- Advanced Oracle Topics
- Distributed Managment
- Product Reviews
- Book Reviews
- News from the Oracle Community
- Oracle User Group Information

Order Now!

YES!, I want to sharpen my Oracle skills. I've checked the subscription plan I'm interested in below

☐ Magazine-Only Subscription Plan...
13 Issues Including One Bonus Issue at $49.95.

☐ Magazine AND Companion Disk Subscription Plan...
13 Issues and Disks Including One Bonus Issue and Disk at $119.95
The Oracle Informant Companion Disk contains source code, support files, examples, utilities, samples, and more!

☐ Oracle Informant Works 1996 CD-ROM $49.95 (Available December 1996)
US residents add $5 shipping and handling. International customers add $15 shipping and handling.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

Country _____ Phone _____

FAX _____ E-Mail _____

Payment Method...

❑ Check (payable to Informant Communications Group) ❑ Purchase Order-- Provide Number _____

❑ Visa ❑ Mastercard ❑ American Express Card Number _____

Expiration Date _____ Signature _____

WEB

To order, mail or fax the adjoining form or call (916) 686-6610 Fax: (916) 686-8497

International rates

Magazine-only
$54.95/year to Canada
$74.95/year to Mexico
$79.95/year to all other countries

Magazine AND Companion Disk Subscriptions
$124.95/year to Canada
$154.95/year to Mexico
$179.95 to all other countries

California Residents add 7¼% sales tax on disk subscription